Erschienen als: Witt, Andreas/Pönninghaus, Jens (2003): Beispiel einer komplexen DTD: DocBook. In: Köhler, Werner/ Wittenbrink, Heinz/Bormann, Carsten.: XML. Berlin: TEIA Lehrbuch Verlag, S. 296-334.

[6] BEISPIEL EINER KOMPLEXEN DTD: DOCBOOK

Nachdem wir in der vorigen Lerneinheit praktisch kennen gelernt haben, wie man XML-Dokumente editieren, mit Parsern auf Wohlgeformtheit prüfen und gegenüber einer DTD validieren kann, werden wir uns in dieser Lerneinheit exemplarisch mit der Benutzung und Anpassung bestehender DTDs befassen. Die DocBook-DTD ist in dieser Hinsicht für uns in zweifacher Weise interessant. Zum einen bietet sie ein hervorragendes Beispiel für die Leistungsfähigkeit von Dokumentgrammatiken, zum anderen ist sie durch ihre weite Verbreitung im Bereich der technischen Dokumentation und für die Erstellung von Fachbüchern auch von unmittelbarer praktischer Bedeutung.

Nach einem kurzen Überblick über die Entstehungsgeschichte von DocBook wird die praktische Verwendung der zur Verfügung stehenden Sprachmittel erläutert. Abschließend wird skizziert, wie der modulare Aufbau der DTD die Anpassung an eigene XML-Anwendungen unter Ausnutzung der Grundstruktur von DocBook unterstützt.

6.1 Einleitung

Die Geschichte von DocBook begann etwa 1991 mit der Bestrebung, ein Austauschformat für die unterschiedlichen UNIX-Dokumentationen zu erstellen. Die Ausrichtung war damit vorgegeben: die Entwicklung der DocBook-DTD bis zum jetzigen Zeitpunkt ist im wesentlichen ein Reifeprozess, der eine akzeptierte SGML- bzw. XML-Anwendung mit einer breiten Palette an Werkzeugen hervorgebracht hat (u. von NOVELL, SUN sowie für die Linux-Dokumentation). Die Werkzeuge decken dabei den gesamten Publikationsprozess von der Dokumenterfassung – auch eingebunden in ein Content-Management-System – bis zur Druck- (RTF, PDF) oder Online-Fassung (HTML, Java-Help etc.) ab. Seit Mitte 1998 wird DocBook bei OASIS weiterentwickelt, seit Version 4 wird eine offizielle XML-Variante unterstützt. Die aktuelle Version ist derzeit (April 2003) DocBook XML V4.2. Offizieller OASIS-Standard bleibt die Version 4.1.2, auf die wir uns im Folgenden immer beziehen werden). Das offizielle Referenzwerk ist das im O'Reilly-Verlag erschienene Buch DOCBOOK: THE DEFINITIVE GUIDE von Norman Walsh und Leonard Muellner DOCBOOK (siehe *Literaturverzeichnis*).

www.oasis-open.org/committees/docbook/xml/

Seite 762

Vielfach wird in der Praxis nicht das „reine" DocBook eingesetzt, sondern es werden die Möglichkeiten zur Einschränkung bzw. Erweiterung der Standard-DTD genutzt, z.B. um hausinterne Stilrichtlinien durchzusetzen und die Autoren bei der Dokumenterstellung durch die Möglichkeiten der Software zu unterstützen.

Im Folgenden werden wir DocBook nicht in seinem vollem Umfang kennen lernen können – zur Zeit sind es etwa 400 Elementtypen –, wohl aber einen repräsentativen Einblick über Struktur und Zusammenwirken der unterschiedlichen Komponenten versuchen.

6.2 Gestaltungsmerkmale der DocBook-DTD

Grundlegendes Gestaltungsmerkmal der DocBook-DTD ist die Trennung in strukturgebendes und inhaltsbezogenes Markup. Den strukturgebenden Teil bilden Elementtypen für Bücher (book), Kapitel (chapter), Abschnitte (z.B. section) usw., während inhaltsbezogenes Markup etwa für Hervorhebungen, Abkürzungen, Zitate, Verweise, Bilder und Grafiken zur Verfügung steht. Darüber hinaus bilden spezielle Elementtypen für die Software-Dokumentation, die zur Beschreibung von Benutzerschnittstellen, Programmiersprachen, Software und Hardware eingesetzt werden können, einen Schwerpunkt von DocBook. Anders als bei SMIL und SVG handelt es sich also um Inhaltsmarkup.

Wir wollen im Folgenden zunächst das strukturgebende Markup der DocBook-DTD anhand eines exemplarischen DocBook-Dokuments vorstellen und daran anschließend anhand eines kleinen Beispiels das inhaltsbezogene Markup erläutern.

Strukturgebendes Markup

Das folgende Beispieldokument für ein Buch enthält einige der wichtigsten Elementtypen der DocBook-DTD:

▶ Beispiel 6.1: Ein exemplarisches DocBook-Dokument

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE book PUBLIC "-//OASIS//DTD DocBook XML V4.1.2//EN"
   "http://www.oasis-open.org/docbook/xml/4.1.2/docbookx.dtd">
<book lang="de">
  <title>Beispiel für die Struktur eines Buches </title>
  <part id="teil-1">
    <title>Teil 1</title>
    <chapter id="kapitel-1-1">
      <title>Überschrift 1-1</title>
      <para>... Hier steht der Text von Kapitel 1 des
            ersten Teils...
      </para>
      <section id="abschnitt-1.1">
        <title>Unterabschnitt</title>
        <para>
         ... Hier steht der Text eines Unterabschnitts von Kapitel 1...
        </para>
      </section>
    </chapter>
  </part>
  <part id="teil-2">
    <title>Teil 2</title>
    <chapter id="kapitel-2-1">
      <title>Überschrift 2-1</title>
      <para>
       ... Hier steht der Text von Kapitel 1 des zweiten Teils...
      </para>
    </chapter>
  </part>
</book>
```

Das im Beispiel dargestellte Buch (book) besteht aus zwei Teilen (part), die jeweils ein Kapitel (chapter) enthalten, wobei das erste Kapitel einen Unterabschnitt (section) enthält. ABBILDUNG 6.1 zeigt eine mögliche Darstellung des Dokuments in HTML.

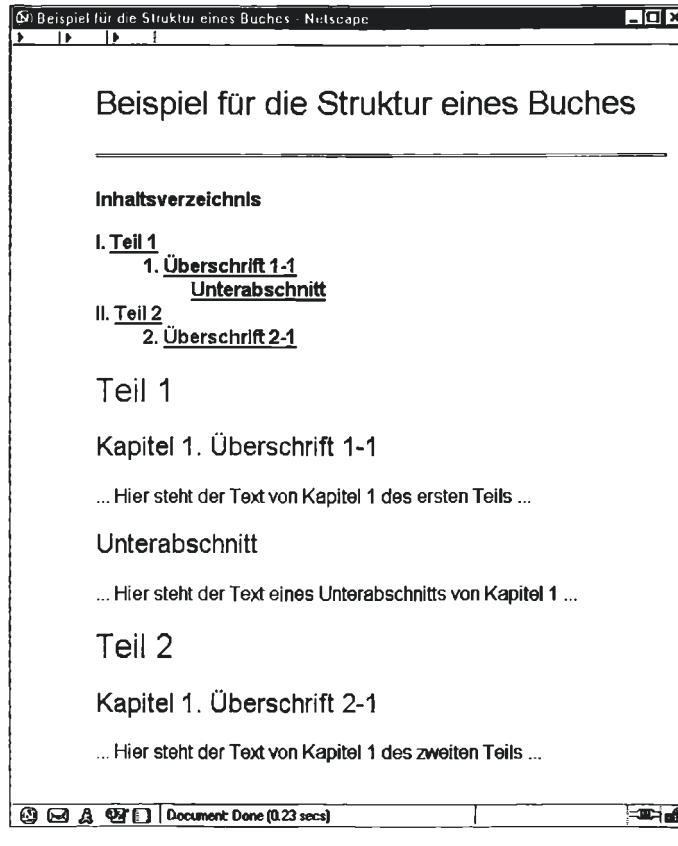

Abbildung 6.1: Ausgabe des DocBook-Dokuments

DocBook selbst enthält keine speziellen layoutbezogenen Elementtypen, die für die visuelle Darstellung eines Dokuments verwendet werden können. Um wie im obigen Beispiel ein DocBook-Dokument in einem Browser darzustellen, muss das DocBook-Dokument zuerst in ein HTML-Dokument transformiert werden. Um XML-Dokumente nach HTML oder in andere Formate zu transformieren, stehen verschiedene

Werkzeuge zur Verfügung, die wir in den kommenden *Lerneinheiten 7 und 8* näher kennen lernen werden. An dieser Stelle genügt es zu sehen, wie eine solche HTML-Darstellung eines DocBook-Dokuments aussehen kann. Im obigen Beispiel sind bei der Transformation nach HTML ein Gesamtinhaltsverzeichnis sowie Inhaltsverzeichnisse für die beiden Teile des Buches automatisch generiert worden. In der Verfügbarkeit solcher automatisierten Generierungsprozesse (vgl. Abschnitt 6.2 – GENERIERBARE DOKUMENTATIONSTEILE) liegt eine der Stärken von DocBook. Betrachten wir nun die einzelnen Elemente des Dokuments etwas genauer.

Seite 336, 395

Seite 296

Das Wurzelelement

Im Verlauf dieser Qualifikationseinheit haben wir bereits gesehen, dass XML-Dokumente eine Baumstruktur repräsentieren und dass es ein spezielles Element gibt, das alle anderen Elemente beinhaltet. Dieses Element wird als Dokumentelement oder Wurzelelement bezeichnet. In XHTML-Dokumenten heißt das Wurzelelement z.B. `html`. In *Lerneinheit 4* wurde bereits kurz angemerkt, dass durch die Angabe einer Dokumenttyp-Deklaration mittels DOCTYPE in der Dokumentinstanz der zu verwendende Typ für ein derartiges Wurzelelement benannt wird, wobei jeder in der DTD definierte Elementtyp diese Funktion übernehmen kann. Hierdurch wird ein starkes Maß an Flexibilität ermöglicht – eine Flexibilität, die allerdings selten genutzt wird.

Seite 208

Bei der Verwendung der DocBook-DTD können für unterschiedliche DocBook-Dokumentinstanzen unterschiedliche Wurzelelemente sinnvoll sein. Wenn aus dem DocBook-Dokument am Ende des Publikationsprozesses ein Buch entstanden sein soll, sollte wie im obigen Beispiel der Elementtyp `book` als Wurzel gewählt werden. Wie bereits der Name der DTD – DocBook – andeutet, kann die Verwendung des Wurzelelementes `book` als der Normalfall angesehen werden. Weitere vollständige und eigenständige Publikationsformen bilden Berichte, Zeitschriften- oder Sammelbandartikel usw., für die der Dokumenttyp `article` vorgesehen ist.

Der Elementtyp `set` kann verwendet werden, wenn mehrere Bücher in einem XML-Dokument zusammengefasst werden sollen. Dieser Fall kann, wenn auch selten, durchaus auftreten. So kann es gewünscht sein, dass mehrere Teile eines Buches auf verschiedene Bände verteilt sein sollen. Es ist auch möglich, Benutzerhandbücher unterschiedlicher, aber dennoch zusammenhängender Programme, z.B. die Referenz eines Office-Paketes, in einer Dokumentinstanz zusammenzufassen.

Die freie Wahl eines Wurzelelementes kann ebenfalls von Vorteil sein, wenn Beiträge verschiedener Autoren zu einem Gesamtdokument zusammengefasst werden sollen. So ist bei umfangreichen Dokumentationen damit zu rechnen, dass bestimmte Abschnitte von verschiedenen Autoren mit unterschiedlichen Qualifikationen verfasst werden. Wollen wir z.B. nur ein Kapitel oder nur einen Abschnitt schreiben und dennoch die DocBook-DTD verwenden, besteht die Möglichkeit, in der Dokumenttyp-Deklaration als Wurzelelement `chapter` anzugeben. Hierdurch können wir auf eine Untermenge der Elementtypdefinitionen der DocBook-DTD zurückgreifen. Dies ermöglicht z.B. den Einsatz von XML-Editoren zur syntaxgesteuerten Texterfassung und die maschinelle Überprüfung der Gültigkeit des Dokuments, d.h. das Parsen der Dokumentinstanz gemäß der DocBook-DTD *(vgl. Lerneinheit 5.3)*.

Seite 257

Generell bleibt jedoch festzuhalten, dass in der Regel nur wenige Elemente als eigenständige Wurzelelemente vorgesehen sind: `set`, `book` und `article`. Sollen nur Teile eines Textes erstellt werden und wird hierfür ein anderes Wurzelelement verwendet, ist darauf zu achten, dass das Gesamtdokument nur eine einzige Dokumenttyp-Deklaration enthalten darf. Hierzu bietet sich die Verwendung externer Entities an, wie sie im Abschnitt **Verwendung externer allgemeiner Entities** dieser Lerneinheit beschrieben sind.

Seite 316

Kapitel und Abschnitte

Nahezu alle Texte können wir folgendermaßen strukturieren: Texte bestehen aus hierarchisch verschachtelten Abschnitten und Unterabschnitten, die mit Überschriften versehen sind. Die Abschnitte sind ihrerseits in Absätze unterteilt. Gedruckte Texte, insbesondere Bücher, sind darüber hinaus noch in Kapitel eingeteilt. Eine derartige Einteilung, d.h. eine Einteilung in Kapitel und verschiedene hierarchisch verschachtelte Abschnitte, ist auch von der DocBook-DTD vorgesehen.

Auch wenn eine derartige Einteilung eines Textes naheliegend, nahezu selbstverständlich erscheinen mag, zeigt sich in der Praxis, dass es andere Einteilungen gibt. So werden beispielsweise in HTML Absätze mit dem Elementtyp `p` direkt markiert, Abschnitte und Unterabschnitte aber nur indirekt gekennzeichnet. Zu welcher Ebene (z.B. Hauptabschnitt, Abschnitt oder Unterabschnitt) ein Absatz gehört, ergibt sich folgendermaßen: Ein Absatz gehört zu der Ebene, zu der die zuletzt verwendete Überschrift gehört (z.B. `h4`, `h6`). Die Abschnitte und die ebenenkennzeichnenden Elemente sind nicht ineinander verschachtelt.

Sehen wir uns jetzt die von der DocBook-DTD zur Definition der Struktur eines Textes zur Verfügung gestellten Elementtypen genauer an: Innerhalb eines Buches (`book`) können insbesondere die Elementtypen `article` und `chapter` Verwendung finden. Typischerweise finden sich Elemente des Typs `article` in Sammelbänden und Elemente des Typs `chapter` in Monographien. Für die weiteren Einheiten eines Buches, vom Inhaltsverzeichnis bis zum Glossar, sind weitere Elementtypen vorgesehen. Unser Dokument aus BEISPIEL 6.1 ist wie gesagt ein BUCH. Nach dem Starttag für das Dokumentelement `book` folgen der Buchtitel (`title`) und ein Element `part`. Das Beispieldokument besteht aus zwei Teilen (engl. „parts") mit den Überschriften „Teil1" und „Teil2", ausgezeichnet durch die jeweiligen `part`-Elemente mit jeweils einem beschreibenden `title`-Element. Die Buchteile sind weiter untergliedert in KAPITEL, ausgezeichnet mit Elementen des Typs `chapter`.

Die weiterführende Ebene der Bücher, sei es ein Artikel, sei es wie in unserem Beispiel eine Anzahl von Kapiteln, wird dann in ABSCHNITTE eingeteilt. Hierfür definiert die DocBook-DTD zwei Möglichkeiten: einerseits kann sich die Hierarchieebene des Abschnittes explizit in der Bezeichnung des die Abschnitte markierenden Elementes widerspiegeln, andererseits können die Abschnitte auch implizit durch die Verschachtelungstiefe gekennzeichnet sein.

Zur expliziten Markierung von Abschnittsebenen sind die fünf Elementtypen `sect1`, `sect2`, `sect3`, `sect4` und `sect5` vorgesehen. Jedem Abschnitt muss mittels des Elementtyps `title` eine Überschrift vorangestellt werden. Den Überschriften folgt entweder Textinhalt, der in Absätze (`para`, nach der englischen Bezeichnung „paragraph" für einen Absatz) eingeteilt wird, Formeln (`equation`), Bilder (`graphic`), Tabellen (`table`, vgl. TABELLEN) oder andere Daten, die zu dem betreffenden Abschnitt gehören.

Sollen Unterabschnitte bei der expliziten Markierung von Abschnittsebenen eingefügt werden, geschieht das in Abhängigkeit von der aktuellen Abschnittsebene. Innerhalb der Ebene `sect1` kann z.B. nur die Ebene `sect2` und innerhalb von `sect4` nur `sect5` eingefügt werden. Die Ebene `sect5` kann nicht weiter unterteilt werden. Wenn dieser explizite Weg der Strukturierung gewählt wird, stehen uns also fünf Hierarchieebenen zur Verfügung. Gestehen wir den Kapiteln den Status einer eigenen Ebene zu, dann besitzen wir – wie in HTML – sechs Gliederungsebenen.

Wird eine implizite Strukturierung vorgenommen, verwenden wir – wie in unserem Beispieldokument – den Elementtyp `section`. Auch die mit diesem Bezeichner markierten Abschnitte müssen mit einer Überschrift (`title`) beginnen und können anschließend die soeben erwähnten Elementtypen für Absätze, Bilder, Tabellen etc. verwenden. Werden die Abschnitte nach der Überschrift weiter unterteilt, werden einfach weitere Elemente des Typs `section` verwendet. Hierbei gibt es keinerlei Beschränkungen der Verschachtelungstiefe. Eine formale Beschränkung dieser Art kann es auch gar nicht geben, da es dafür erforderlich wäre, den Elementtyp `section` mehrfach und unterschiedlich zu definieren. Dies ermöglicht die Semantik der XML-DTDs nicht. Neuere Sprachen zur Definition von Dokumenttypen erlauben auch derartige Definitionen von Elementtypen *(vgl. Lerneinheit 10).*

Seite 509

Erstellen wir einen Text mit der DocBook-DTD, können wir also wählen, welche Art der Unterteilung strukturierter Abschnitte wir verwenden. Diese Wahl muss am Beginn eines Kapitels oder Artikels erfolgen, d.h. wir haben die Alternativen `sect1` (explizite Markierung) und `section` (implizite Markierung). Haben wir uns entschieden, müssen wir den Weg weitergehen, d.h. es ist weder möglich, innerhalb der Ebene `sect5` ein Element `section` einzufügen, noch kann innerhalb der Umgebung `section` ein Element `sect1` verwendet werden.

TABELLE 6.1 zeigt noch einmal die wichtigsten Elementtypen von DocBook für das strukturelle Markup.

Tabelle 6.1: Übersichtstabelle: Die wichtigsten Elementtypen zum strukturellen Markup

Elementtyp	Beschreibung
`book`	Das Wurzelelement eines Buchs
`article`	Das Wurzelelement eines Berichts, Artikels oder ähnlicher Werke
`chapter`	Ein Kapitel eines Buchs
`sect1` bis `sect5`	Abschnitte von Kapiteln oder Artikeln entsprechend ihrer Hierarchieebene
`section`	Abschnitte von Kapiteln oder Artikeln. Die Elemente vom Typ `section` können weitere Elemente dieses Typs enthalten. Die Hierarchieebene ergibt sich hierbei durch die Verschachtelungstiefe.
`para`	Absätze

Weitere strukturierende Elementtypen

Weitere wichtige, die Texte strukturierende Elementtypen erlauben die Erstellung von Querverweisen, Listen und Tabellen. Allerdings lässt sich darüber streiten, ob solche Elemente tatsächlich so eindeutig strukturgebend sind oder ob sie nicht vielmehr bereits als inhaltsorientiert klassifiziert werden müssen. Während die Angaben über Absätze, Abschnitte, Überschriften, Kapitel usw. eindeutig als strukturell aufzufassen sind und die Elementtypen zur Markierung spezieller technischer Einheiten und Vorgänge (z.B. Menüeinträge, Tastaturkürzel etc.) zweifelsohne als inhaltsorientiert bezeichnet werden müssen, bilden Querverweise, Listen und Tabellen eine Zwischenschicht, die sowohl strukturierende als auch inhaltsbezogene Merkmale besitzen.

Wie bereits gesagt, können wir nur wenige aus der Vielzahl der in der DocBook-DTD definierten Elementtypen behandeln. Die in der Doc-

Book-DTD verwendeten Sprachmittel zur Modellierung von Tabellen werden wir später betrachten. Sie werden im Abschnitt TABELLEN beschrieben. Hier wollen wir kurz auf die Elementtypen für Listen und Querverweise eingehen.

Listen

Für die Erstellung von Listen stehen verschiedene, zum Teil auch stärker inhaltsbezogene Elementtypen zur Verfügung. Die in der DocBook-DTD vorkommenden Listenarten sind: `calloutlist`, `itemizedlist`, `orderedlist`, `segmentedlist`, `simplelist`, `variablelist`. Wir wollen uns nun einige davon etwas näher ansehen, wobei die getroffene Auswahl nichts über die Bedeutung der einzelnen Elementtypen aussagen soll, sondern nur aus Platzgründen beschränkt bleibt.

In einer einfachen Liste des Typs `simplelist` muss mindestens ein Element vom Typ `member` auftreten. Dieses enthält den jeweiligen Listeneintrag. Dieser Eintrag kann nicht stark strukturiert sein, d. h. beispielsweise, dass innerhalb des Elements `member` keine weiteren Absätze auftreten dürfen. Derartige einfache Listen dürfen nicht ineinander verschachtelt sein. BEISPIEL 6.2 zeigt die Grundstruktur einer einfachen Liste.

▶ Beispiel 6.2: Einfache Liste (simplelist)

```
<simplelist>
  <member>Listeneintrag 1 </member>
...
  <member>Listeneintrag n </member>
</simplelist>
```

Die Elementtypen `itemizedlist` und `orderedlist` sind vergleichbar mit den ungeordneten und den geordneten Listen in HTML. Innerhalb dieser Listen muss mindestens ein Element vom Typ `listitem` auftreten. Innerhalb dieser Listeneinträge können sowohl strukturierte längere Textpassagen als auch weitere Listen auftreten, wie in BEISPIEL 6.3 angedeutet.

► **Beispiel 6.3: Ungeordnete Listen (itemizedlist) und geordnete Listen (orderedlist)**

```
<itemizedlist>
  <listitem>
    <para>Listeneintrag 1</para>
  </listitem>
...
  <listitem>
    <para>Listeneintrag n</para>
  </listitem>
</itemizedlist>

...

<orderedlist>
  <listitem>
    <para>Listeneintrag 1</para>
  </listitem>
...
  <listitem>
    <para>Listeneintrag m</para>
  </listitem>
</orderedlist>
```

Querverweise

Ebenfalls vergleichbar mit HTML ist die Verwendung von Verknüpfungen und Querverweisen, wobei die DocBook-DTD auch hier wesentlich mächtigere Möglichkeiten zur Verfügung stellt. In der einfachsten Variante wird ein Querverweis mit einem Element vom Typ `link` gekennzeichnet. Dieses Element muss einen Wert für das Attribut `linkend` enthalten. Das Attribut ist vom Typ `IDREF` und gibt das Ziel des Querverweises in Form eines eindeutigen Elementbezeichners an. Durch das optionale Attribut `endterm`, ebenfalls vom Typ `IDREF`, kann der Inhalt des angegebenen Elements als Text des Links ausgegeben werden:

```
<link linkend="kapitel1" endterm="kapitel1.titel"/>
```

Für die Verknüpfung mit Ressourcen außerhalb des aktuellen Dokuments steht in DocBook unter anderem der Elementtyp `ulink` zur Verfügung. Im Attribut `url` kann ein beliebiger URL als Ziel für den Verweis spezifiziert werden. Neben den Elementtypen `link` und `ulink` gibt es in DocBook noch andere Elementtypen für Querverweise (`xref`, `olink`), die hier zumindest genannt sind, auf die wir aber an dieser Stelle nicht näher eingehen wollen.

Weitere inhaltsbezogene Elementtypen

Auch für die Auszeichnung von Wörtern, Wortgruppen, Sätzen und Satzgruppen gibt es in der DocBook-DTD eine Vielzahl von Elementtypen, mit denen unter anderem Hervorhebungen (`emphasis`), Anführungen/Zitate (`quote`), Abkürzungen (`abbrev`) oder Akronyme (`acronym`) gekennzeichnet werden können. Einfache Elementtypen für Präsentationsmarkup, wie z.B. `b` für „fett" oder `i` für „kursiv" in HTML, gibt es nicht, da es sich bei DokBook um ein Vokabular zur inhaltlichen Auszeichnung von Dokumenten handelt. Es lässt sich für diese Zwecke aber das Attribut `role` des Elementtyps `emphasis` einsetzen:

```
<para>
  So kann Text als <emphasis role="italics"
    >kursiv</emphasis> oder <emphasis role="bold"
    >fett</emphasis> ausgezeichnet werden.
</para>
```

Der Wert des Attributes `role` ist durch DocBook nicht festgelegt. Es bleibt einem Anwendungsprogramm überlassen, diese Werte zu interpretieren. Für die Transformation nach HTML könnten die Werte des Attributs z.B. in ein Attribut `class` übernommen werden. Die Darstellungsattribute würden dann in einem zugehörigen Stylesheet festgelegt:

```
<p>
  So kann Text als <span class="italics">kursiv</span>
    oder
  <span class="bold">fett</span> ausgezeichnet werden.
</p>
```

Generierbare Dokumentationsteile

Ein großer Vorteil bei der Verwendung von strukturgebendem Markup besteht darin, dass durch die maschinelle Verarbeitung derartiger Texte Nutzen aus der Zusatzinformation gezogen werden kann. Im einfachsten Fall besteht der Nutzen z.B. schlicht darin, dass ein WWW-Browser den Text, der durch ein HTML-Element des Typs `h1` gekennzeichnet ist, größer anzeigt als den Text, der als `h5` gekennzeichnet wurde. Die Interpretation der Elemente für Listeneinträge in HTML (`li`) ist schon komplexer, da in diesem Fall der Browser beachten muss, ob das Element innerhalb eines Elements `ol` (für geordnete Listen) oder innerhalb eines Elements `ul` (für ungeordnete Listen) auftritt. Zusätzlich muss für Einträge geordneter Listen auch noch deren Position innerhalb der Liste ermittelt und in geeigneter Form dargestellt werden.

Wenn XML-Dokumente angezeigt oder ausgedruckt werden sollen, muss die intendierte Bedeutung der Verwendung aller Elemente explizit gemacht werden. Hierfür wurden so genannte STYLE LANGUAGES entwickelt, wobei diese Sprachen mehr oder weniger komplex sein können. Die speziell im Zusammenhang mit XML entwickelten derartigen Sprachen XSLT und XSL-FO werden in den *Lerneinheiten 7 und 8* vorgestellt. Diese Sprachen sind äußerst komplex. Ein Beispiel für eine relativ einfache STYLE LANGUAGE sind die so genannten CASCADING STYLE SHEETS (CSS), die gemeinsam mit HTML-, aber auch mit XML-Dokumenten verwendet werden können.

Seite 336, 395

Vor ihrer Veröffentlichung (z.B. als Buch oder im WWW) unterliegen XML-Dokumente oftmals einem Verarbeitungsprozess mit zahlreichen Teilschritten. Die Verarbeitung ist dabei keineswegs auf die Festlegung von Darstellungseigenschaften beschränkt. Eine der wichtigsten weitergehenden Formen der Verarbeitung von XML-Dokumenten bildet die automatische Erstellung bzw. Generierung von Teilen der Texte. Insbesondere können Verzeichnisse generiert werden.

Zu Texten, die konform zur DocBook-DTD verfasst wurden, können sehr viele Verzeichnisse automatisch erstellt werden, z.B. das Inhaltsverzeichnis, ein Abbildungsverzeichnis, ein Tabellenverzeichnis und ein Stichwortverzeichnis. Dieser Prozess basiert darauf, dass die in die Verzeichnisse zu übernehmenden Textteile korrekt ausgezeichnet wurden. Bei der Generierung der Verzeichnisse werden alle Elemente daraufhin überprüft, ob sie in das Verzeichnis aufgenommen werden sollen. So werden

bei der automatischen Erstellung eines Inhaltsverzeichnisses für ein Buch alle Überschriften (`title`) zusammengetragen, in das generierte Verzeichnis kopiert und mit einer Seitenangabe versehen. Analog hierzu verlaufen alle weiteren Verzeichnisgenerierungsprozesse. In unserem Beispieldokument (BEISPIEL 6.1) werden die Elementtypen `part`, `chapter` und `section` bei der Generierung des Inhaltverzeichnisses berücksichtigt.

Inhaltsbezogenes Markup

Die größte Gruppe der DocBook-Elementtypen dient der inhaltsorientierten Auszeichnung der Daten. Darunter verstehen wir die Annotierung der Daten in Bezug zu dem im Dokument beschriebenen Gegenstandsbereich, also zum Beispiel die Auszeichnung von Produktnamen, Menüeinträgen oder Inventarisierungsnummern – im Gegensatz zur strukturellen Auszeichnung, die weitgehend frei von Bezügen zum eigentlichen Gegenstand des Textes ist. Im Folgenden werden wir exemplarisch einige Gruppen dieser inhaltsorientierten Elementtypen betrachten. Die exakte Semantik und damit die korrekte Verwendungsweise ist immer gesondert in entsprechenden Richtlinien („STYLE GUIDES"), d. h. detaillierten Beschreibungen des DTD-Vokabulars, zu vereinbaren. Die DocBook-Referenz DOCBOOK (siehe *Literaturverzeichnis*) gibt hierbei erste Hinweise, bietet aber durch die Vielzahl an Elementtypen oft auch äquivalente Möglichkeiten bei der Verwendung, so dass in der Praxis darüber hinausgehende spezifische Festlegungen getroffen werden müssen.

Seite 762

Der prototypische Anwendungsbereich von DocBook ist die Programmdokumentation und Erstellung von Nutzerhandbüchern, daher stehen für diesen Kontext zahlreiche Elementtypen zur Verfügung. Dies reicht von Elementtypen zum Markieren von Dateinamen (`filename`) bis zur Repräsentation einer Methodenzusammenfassung (`methodsynopsis`) einer objektorientierten Programmiersprache. Deshalb werden wir im Folgenden exemplarisch den Anwendungsbereich der Dokumentation einer grafischen Benutzerschnittstelle (engl. „graphical user interface", GUI) darstellen sowie einige andere gebräuchliche Elementtypen benennen.

Anwendungsbereich GUI-Dokumentation

Die Hauptlast der Dokumentation entfällt häufig auf die konsistente Darstellung der Nutzerinteraktion, mit Systemantworten, Verweisen auf Menüeinträge sowie Tastaturkürzeln und Programmaufrufen. DocBook bietet dafür vor allem die in TABELLE 6.2 aufgeführten Elementtypen an.

Elementtyp	Beschreibung
`guimenu`	Bezeichnet ein Hauptmenü und enthält oft ein Tastaturkürzel (`accel`)
`guisubmenu`	Ein Untermenü, das Einträge oder weitere Menüs enthält
`guimenuitem`	Menüpunkt
`menuchoice`	Auswahlmöglichkeit, etwa bei Selektionsmenüs oder bei alternativem Zugriff über Tastaturkürzel
`accel`	Lokales Tastaturkürzel innerhalb des Menüs – häufig der Anfangsbuchstabe des Menüeintrags
`shortcut`	Tastaturkürzel für Menüpunkt
`keycap`	einzelner Tastendruck
`keycombo`	Tastenkombination, z.B. sequentiell oder zeitgleich
`mousebutton`	symbolischer Name einer Maustaste

Tabelle 6.2: Elementtypen für Menüeinträge und Tastatureingaben

BEISPIEL 6.4 zeigt das Markup für die Tastatursequenz im Editor Emacs zum Einfügen eines XML-Elements innerhalb des psgml-Modus.

Beispiel 6.4: Emacs-Tastatursequenz

```
<para>Um ein neues Element einzufügen, drücken Sie bitte
    folgende Tastenkombination:
  <keycombo action="seq">
    <keycombo action="simul">
      <keycap>Strg</keycap>
      <keycap>c</keycap>
    </keycombo>
    <keycombo action="simul">
      <keycap>Strg</keycap>
      <keycap>e</keycap>
    </keycombo>
  </keycombo>
</para>
```

Eine mögliche Darstellung dieses Beispiels auf einem Ausgabemedium könnte dann wie folgt aussehen:

Um ein neues Element einzufügen, drücken Sie bitte folgende Tastenkombination:
Strg+c Strg+e

Das inhaltsbezogene Markup legt hierbei nur fest, welche einzelnen Elemente überhaupt unterschieden werden, aber nicht, wie letzlich ihre Darstellung auf einem Ausgabemedium erfolgt. Eine weitere mögliche Darstellung kann z. B. andere Verbindungszeichen für die Tastatursequenz realisieren und auch andere Tastenbezeichnungen verwenden. Die Standarddokumentation von Emacs verwendet beispielsweise als Verbindungszeichen statt des „+" ein „-" und für die Taste „Strg" die Bezeichnung „C", so dass dort die Tastenkombination „Strg+c Strg+e" als „C-c C-e" dargestellt wird. Wie Markup für die Darstellung auf einem Ausgabemedium konkret umgesetzt wird, werden wir in den *Lerneinheiten 7 und 8* näher betrachten.

Seite 336, 395

Als Beispiel für das Markup einer Mauseingabe verwenden wir eine alternative Eingabemöglichkeit zu der Tastatursequenz im Beispiel oben. Ein Element lässt sich auch durch Aufruf eines Menüeintrags einfügen. Für die Annotierung wird der Elementtyp `mousebutton` verwendet. Die grundlegende Form einer Menü-Darstellung verdeutlicht BEISPIEL 6.5.

▶ Beispiel 6.5: Auswahl über einen Menüeintrag

```
<para>Ein Element kann auch über die psgml-Erweiterung
    der Menüleiste eingefügt werden: Dazu wählen Sie mit
    der Maus
(<mousebutton>linke Maustaste</mousebutton>)im Hauptmenü
    den Eintrag
<guimenu>Markup</guimenu> und dann den Untermenüeintrag
<guimenuitem>Insert Element</guimenuitem>.</para>
```

Die Umsetzung dieses Beispiels könnte wie folgt aussehen:

Ein Element kann auch über die psgml-Erweiterung der Menüleiste eingefügt werden: Dazu wählen Sie mit der Maus *(linke Maustaste)* im Hauptmenü den Eintrag ***Markup*** und dann den Untermenüeintrag ***Insert Element***.

BEISPIEL 6.6 veranschaulicht schließlich die Verbindung der unterschiedlichen Elemente zu einer kompakteren Beschreibung:

Beispiel 6.6: Kompakte Darstellung

```
<para>
Sie können einen markierten Bereich mit einem neuen
  Element durch Eingabe von
<menuchoice>
  <shortcut>
    <keycombo action="seq"><keysym>Strg+c</keysym>
      <keysym>Strg+r</keysym></keycombo>
  </shortcut>
  <guimenu>Markup</guimenu>
  <guimenuitem>Tag Region</guimenuitem>
</menuchoice> auszeichnen.
</para>
```

Nachfolgend wieder eine mögliche Darstellung dieses Beispiels:

Sie können einen markierten Bereich mit einem neuen Element durch Eingabe von ***Markup->TagRegion (Strg+c Strg+r)*** auszeichnen.

Die Darstellung der Tastatursequenz wird hier also durch die Vergabe symbolischer Werte (`Strg+c` bzw. `Strg+r` als Inhalt eines Elements vom Typ `keysym` repräsentieren jeweils die entsprechende Tastenkombination) verkürzt und als `shortcut` des Menüeintrags dargestellt.

Hinweise, Kommentare, Warnungen, Tipps

Sehr häufig ist ein Textabschnitt mit einer explizit zu benennenden Intention des Autors besetzt, z.B. im Falle von Warnungen, erläuternden Hinweisen oder Tipps. DocBook hält eine Reihe von entsprechenden Elementtypen bereit, mit denen der Autor zum einen die logische Struktur seines Texts verdeutlichen kann, die zum anderen aber auch eine gezielte Weiterverarbeitung, insbesondere in Hinblick auf die Präsentation des Texts, ermöglichen. So sollen die durch Elemente dieser Typen ausgezeichneten Inhalte häufig als Marginalien zur Ergänzung des Haupttextes gesetzt werden.

DocBook sieht die in TABELLE 6.3 aufgeführten „Ermahnungen" (engl. ADMONITIONS) vor, wobei die Semantik nicht strikt vorgegeben ist und in einer hausinternen Stilregel festgelegt werden sollte.

Elementtyp	Beschreibung (typische Verwendung)
`caution`	Gefahr einer Beschädigung
`important`	Wichtiger Aspekt
`note`	einfache Anmerkung
`tip`	Ein Tipp, z.B. zu geschickterem Vorgehen
`warning`	Gefahr eines Personenschadens

Tabelle 6.3: Warnungen, Tipps etc.

Wie bei allen Elementtypen in DocBook lässt sich die anwendungsspezifische Bedeutung bei der Verwendung in einer Dokumentinstanz mit entsprechenden Rollenattributen feiner abstufen. Eine geeignete Weiterverarbeitung kann dann z.B. `tip`-Elemente, die sich nur auf ein bestimmtes Betriebssystem beziehen, verarbeiten und solche zu anderen Zielplattformen unterdrücken. So ist es durch die Attribuierung der Elemente möglich, innerhalb eines logischen Dokuments unterschiedliche Profile einer Dokumentation zu halten.

Gebräuchliche Elementtypen

Ein fester Grundbestandteil an inhaltsorientiertem Markup lässt sich nur schwer definieren, die hier getroffene Auswahl ist an der Programmdokumentation – z.B. für die grafische Benutzeroberfläche KDE – orientiert. TABELLE 6.4 zeigt einige weitere Elementtypen von DocBook, die in derartigen Dokumenten häufig auftreten.

Elementtyp	Beschreibung
`application`	Ein Anwendungsprogramm
`command`	Ein Programmaufruf; z.B. `'mkdir'`
`option`	Eine Option eines Programms; z.B. `'--help'`
`filename`	Datei- oder Verzeichnisname; z.B. `'../Texte/Einführung.txt'`
`userinput`	vom Nutzer zu tätigende bzw. getätigte Eingabe
`replaceable`	Überschreibbare Vorgabe (z.B. ein vorgegebener Dateiname)
`errorcode`	Fehlercode, z.B. `'404'` in der Antwort eines Webservers auf eine Anfrage
`screen`	Bildschirmdarstellung; z.B. einer Terminalsitzung
`guibutton`	Eine Schaltfläche; z.B. `'Abbrechen'`
`guiicon`	Beschreibung oder Name eines Symbols, z.B. `'Lupe'`
`guilabel`	Beschriftung; z.B. `'Name:'` vor einem Eingabefeld

Tabelle 6.4: Gebräuchliche Elementtypen für Programmdokumentationen

6.3 Externe Entities

Seite 208

Die Verwendung von Entities wurde bereits in *Lerneinheit 4* vorgestellt. Wir haben gesehen, dass es möglich ist, wiederverwendbare Dokumentbestandteile als Entities zu definieren. Hierbei wird grundsätzlich zwischen allgemeinen (generellen) Entities und Parameter-Entities unterschieden. Die Definition erfolgt jeweils in der DTD, die Einbindung der Dokumentbestandteile (Entity-Referenzen) erfolgt im Falle allgemeiner Entities in der Dokumentinstanz, bei der Verwendung von Parameter-Entities hingegen in der DTD.

Parameter-Entities bilden im Zusammenhang mit MARKED SECTIONS ein sehr wichtiges Strukturierungselement in komplexen DTDs. Hierfür werden Klassen strukturell ähnlicher Elemente und Attribute gebildet und zusammengefasst. Diese Klassen werden dann verschiedenen Parameter-Entities zugeordnet. Von diesen Entities wird in den DTDs bei der Deklaration von Elementtypen umfangreich Gebrauch gemacht. Dieses Vorgehen erlaubt zum einen ein besseres Verständnis der DTD, zum anderen verbessert es die Erweiterungsmöglichkeiten.

Allgemeine Entities werden ebenfalls in der DTD deklariert, werden allerdings in der Dokumentinstanz verwendet. Ihre Verwendung kann aus verschiedenen Gründen erwünscht sein, z.B. um häufig wiederkehrende Textblöcke abkürzen zu können oder um (Sonder-)Zeichen in die Texte einfügen zu können, die nicht über die Tastatur verfügbar sind oder für die es keine Entsprechung in der gewählten Zeichenkodierung für das XML-Dokument gibt (vgl. dazu auch *Lerneinheit 2).*

Seite 82

Die Verwendung von Entities erlaubt es darüber hinaus, dass sowohl die DTD als auch die Dokumentinstanz, die jeweils eine logische Einheit bilden, auf verschiedene Dateien verteilt werden können. Derartige Entities werden als externe Entities bezeichnet. Es gibt sowohl externe allgemeine Entities als auch externe Parameter-Entities.

Anhand zweier Beispiele werden wir die Verwendungsweise von externen allgemeinen Entities und von externen Parameter-Entities nun kennen lernen.

Verwendung externer allgemeiner Entities

Externe allgemeine Entities erlauben z.B., dass die einzelnen Kapitel eines Buches auf verschiedene Dateien verteilt werden. BEISPIEL 6.7 zeigt die Aufteilung des Beispielsdokuments auf die Dateien `buch.xml`, `kapitel1.xml` und `kapitel2.xml`. Die externen Entities werden dazu im so genannten internen DTD-Subset deklariert.

▶ Beispiel 6.7: Verwendung von Enitities und Entity-Referenzen in DocBook-Dokumenten (1)

- buch.xml:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE book PUBLIC "-//OASIS//DTD DocBook XML V4.1.2//EN"
          "http://www.oasis-open.org/docbook/xml/4.1.2/docbookx.dtd" [
  <!ENTITY chap1 SYSTEM "kapitel1.xml">
  <!ENTITY chap2 SYSTEM "kapitel2.xml">
]>
<book lang="de">
  <title>Beispiel für die Struktur eines Buches </title>
  <part id="teil-1">
    <title>Teil 1</title>
    &chap1;
  </part>
  <part id="teil-2">
    <title>Teil 2</title>
    &chap2;
  </part>
</book>
```

- kapitel1.xml:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<chapter id="kapitel-1">
  <title>Überschrift 1</title>
  <para>... Hier steht der Text von Kapitel 1...</para>
  <section id="abschnitt-1.1">
    <title>Unterabschnitt</title>
    <para>
    ... Hier steht der Text eines Unterabschnitts von
      Kapitel 1...
    </para>
  </section>
</chapter>
```

- kapitel2.xml:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<chapter id="kapitel-2">
  <title>Überschrift 2</title>
    <para>... Hier steht der Text von Kapitel 2...</para>
</chapter>
```

In diesem Beispiel werden in der Datei `buch.xml` zwei externe Entities (CHAP1 und CHAP2) mit den Systembezeichnern `kapitel1.xml` und `kapitel2.xml` deklariert. Die entsprechenden Dateien `kapitel1.xml` und `kapitel2.xml` dürfen keine eigene Dokumenttyp-Deklaration enthalten, da sonst nach Auflösung der Entity-Referenz im Hauptdokument (`buch.xml`) mehrere Dokumenttyp-Deklarationen stünden. Um die Dateien dennoch als DocBook-Instanzen bearbeiten zu können (um z.B. entsprechende Werkzeuge zum Editieren und Validieren einzusetzen), bildet man einfach ein kleines XML-Dokument (z.B. `chap1.xml`), das bloß aus der Dokumenttyp-Deklaration, der Entity-Deklaration und der Entity-Referenz besteht (siehe BEISPIEL 6.8).

► **Beispiel 6.8: Verwendung von Enitities und Entity-Referenzen in DocBook-Dokumenten (2)**

– chap1.xml:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE chapter PUBLIC "-//OASIS//DTD DocBook XML V4.1.2//EN"
          "http://www.oasis-open.org/docbook/xml/4.1.2/docbookx.dtd" [
  <!ENTITY chap1 SYSTEM "kapitel1.xml">
]>
&chap1;
```

Verwendung externer Parameter-Entities

Externe Parameter-Entities unterscheiden sich aus der Anwendungsperspektive betrachtet von den externen allgemeinen Entities dadurch, dass sie nicht nur in der DTD deklariert werden, sondern auch nur in der DTD verwendet werden (dürfen). Die Syntax der Entities unterscheidet sich, wie in *Lerneinheit 4* gesehen, ebenfalls nur wenig: Referenzen auf Parameter-Entities werden bei der Verwendung mit einem Prozentzeichen „%" an Stelle eines Kaufmanns-Und „&" gekennzeichnet. Bei der Deklaration wird ein Prozentzeichen zwischen das Schlüsselwort ENTITY und den Namen der Entity aufgenommen. Die DocBook-DTD verwendet eine Vielzahl externer Parameter-Entities, was wir unter anderem schon daran sehen, dass die DTD auf viele Dateien verteilt ist.

Seite 208

In der DocBook-DTD werden allerdings auch hunderte allgemeiner Entities eingebunden, deren Deklarationen ihrerseits auf verschiedene externe Entities verteilt sind. Diese externen Entities enthalten spezielle Zeichen aus den so genannten ISO-Zeichensätzen. Im einzelnen untergliedert sich diese Menge externer Entities unter anderem in `iso-cyr1.ent` und `iso-cyr2.ent` (für russische und nicht-russische kyrillische Zeichen), `iso-grk1.ent` bis `iso-grk4.ent` (für verschiedene griechische Zeichensätze), `iso-lat1.ent` und `iso-lat2.ent` (für west-

und osteuropäische Schriften, die auf der lateinischen Schrift basieren, z. B. die deutsche Schrift inklusive ä, ö, ü, ß etc.) sowie Zeichensätze für mathematische und technische Zwecke (`iso-num.ent`, `iso-pub.ent`, `iso-tech.ent`, `iso-amsa.ent`, `iso-amsb.ent`, `iso-amsc.ent`, `iso-amsn.ent`, `iso-amso.ent` und `iso-amsr.ent`). Diese Entity-Deklarationen sind integraler Bestandteil der DocBook-DTD und müssen nicht separat eingebunden werden.

Tabellen

Als Beispiel für eine externe Parameter-Entity betrachten wir nun das in die DocBook-DTD integrierte CALS-Tabellenmodell. CALS ist eine Initiative des amerikanischen Verteidigungsministeriums und stand ursprünglich für „Computer Aided Acquisition and Logistic Support". Mittlerweile wird es als Akronym für „Continuous Acquisition and Life-Cycle Support" verwendet. In dieser Initiative wurden zahlreiche Standards für die technische Dokumentation entwickelt, die vor allem für das US-Militär von Bedeutung sind. In dem Zusammenhang ist auch ein Modell für die (ursprünglich SGML-basierte) Notation von Tabellen entwickelt worden. Das im CALS-Standard MIL-HDBK-28001 beschriebene Modell für Tabellen verwendet die XML-Syntax, gilt allgemein als ausgereift und anerkannt und stand auch bei der Implementierung des Tabellenmodells für HTML Pate. Das CALS-Tabellenmodell ist als externe Entity integraler Bestandteil der DocBook-DTD. Es ist allerdings auch möglich, die externe Entity, die das DTD-Modul zur Definition der CALS-Tabellen enthält, in andere DTDs zu integrieren. Dies geschieht in analoger Weise zur Einbindung der allgemeinen Entities, die wir am Beispiel der Verwendung externer allgemeiner Entities betrachtet haben.

Das CALS-Tabellenmodell erlaubt es, nahezu alle vorstellbaren Tabellen zu repräsentieren. Wir wollen uns dieses Modell anhand eines kleinen Beispiels aus der Autobranche genauer ansehen. TABELLE 6.5 zeigt die Zuordnung von Felgen und Reifengrößen für drei verschiedene PKW-Typen.

Leistung Typ	37KW	55KW	82KW
Stahlfelgen Notrad	5JX13	51/2JX13	51/2J13
	31/2Jx14		
Leichtmetall	5Jx13	6Jx14	6Jx14
Reifen, radial, schlauchlos Notrad	175/70SR1380	185/60HR14080H	185/60HR1480H
	T105/70D 14 oder T105/70 R 14		

Tabelle 6.5: Radgrößen

Die Grundidee bei CALS-Tabellen (und in ähnlicher Form findet sich dieses Konzept auch in HTML) ist die lineare Anordnung der Tabellenzellen als Folge von Zeilen (`row`), in denen Elemente für die einzelnen Zellen (`entry`) – nach Spalten – von links nach rechts geordnet aufgeführt werden. Die Formatierung einer Tabelle erfolgt dann zeilenweise, wobei in jeder Zeile nacheinander die einzelnen Zellen verarbeitet werden. Spezielle Attribute erlauben es, aufeinanderfolgende Spalten bzw. Zeilen zusammenzufassen. CALS-Tabellen unterscheiden sich von HTML-Tabellen unter anderem durch eine genauere Spezifikation der Zellengruppierung. Dadurch sind diese Tabellen in der XML-Notation aber auch häufig etwas komplizierter und vom menschlichen Betrachter nicht mehr auf den ersten Blick zu durchschauen.

BEISPIEL 6.9 zeigt, wie TABELLE 6.5 realisiert wurde (die Nummern am Rand verweisen auf die im Anschluss gegebenen Erläuterungen).

▶ Beispiel 6.9: Eine CALS-Tabelle

```
<table frame="all">
  <title>Räder</title>
    <tgroup cols="4" align="left" colsep="1"          1)
       rowsep="1">
    <colspec colnum="2" colname="c2"/>                2)
    <colspec colnum="4" colname="c4"/>
    <thead valign="top">                              3)
      <row>
        <entry>Leistung Typ</entry>
        <entry>37KW</entry>
        <entry>55KW</entry>
        <entry>82KW</entry>
```

```
      </row>
    </thead>

    <tbody valign="top">                                          4)
      <row>
        <entry morerows="1">Stahlfelgen Notrad</entry>  5)
        <entry align="center">5JX13</entry>
        <entry>51/2JX13</entry>
        <entry>51/2J13</entry>
      </row>
      <row>
        <entry namest="c2" nameend="c4" align="center"  6)
             >31/2Jx14</entry>
      </row>
      <row>
        <entry>Leichtmetall</entry>
        <entry align="center">5Jxl3</entry>
        <entry>6Jx14</entry>
        <entry>6Jx14</entry>
      </row>
      <row>
        <entry morerows="1">Reifen, radial, schlauchlos
           Notrad</entry>
        <entry>175/70SR1380</entry>
        <entry>185/60HR14080H</entry>
        <entry>185/60HR1480H</entry>
      </row>
      <row>
        <entry namest="c2"
               nameend="c4"
               align="center">T105/70D 14 oder
                  T105/70 R 14</entry>
      </row>
    </tbody>
  </tgroup>
</table>
```

1) Definition eines Tabellenkörpers mit vier Spalten (Attribut `cols`). Das Attribut `align` gibt die horizontale Ausrichtung des Inhalts in jeder Tabellenzelle an. In diesem Fall wurde mit „`left`" eine Ausrichtung an der linken Seite (in west-europäischen Schriftsystemen) gewählt. Eine zentrierte Ausrichtung kann mit der Angabe `align="center"` erreicht werden, eine rechtsbündige Darstellung bewirkt die Angabe `align= "right"`. Seltener verwendet werden die Werte „`justify`" für Blocksatz und „`char`" für eine Ausrichtung an einem bestimmten Zeichen, das dann in einem zusätzlichen Attribut `char` spezifiziert werden muss.
   Die beiden verbleibenden Attribute steuern das Vorhandensein von Trennlinien zwischen Tabellenzellen. Die erlaubten Werte hier sind „`0`" und „`1`" . Hier wurde mit `colsep="1"` und `rowsep="1"` bestimmt, dass sowohl Spalten (`colsep`, für „column separator") als auch Zeilen (`rowsep`, für „row separator") mit einer Linie begrenzt sein sollen.
2) Benennung der Spalten 2 und 4 mit Namen c2 bzw. c4; diese Benennungen brauchen wir im Folgenden für Verweise auf diese Spalten.
3) Der Bereich `thead` definiert den Tabellenkopf, bestehend aus einer Zeile mit vier Zellen. Die Zeile wird durch ein Element des Typs `row` definiert. Jede Zelle wird durch ein Element des Typs `entry` definiert. Das Attribut `valign` spezifiziert die vertikale Ausrichtung des Inhalts innerhalb einer jeden Tabellenzelle. Die Angabe „`top`" dient zur Ausrichtung an der oberen Kante einer Zelle, mit „`middle`" und „`bottom`" lässt sich der Zelleninhalt in der Mitte bzw. unten anordnen.
4) Der Bereich `tbody` definiert die Nutzdaten; primär organisiert in Zeilen mit variabler horizontaler und vertikaler Ausdehnung der Inhalte.
5) In der ersten Zeile wird die erste Zelle mit der direkt darunter liegenden Zelle verschmolzen (`morerows="1"`). Dies vertikale Verschmelzen wird `STRADDLING` genannt. Der Wert des Attributs `morerows` gibt an, wie viele Zeilen verschmolzen werden. Die verschmolzenen Zellen werden im Folgenden nicht mehr angegeben. Von den vier `entry`-Elementen der Folgezeile muss folglich das erste Element weggelassen werden.

6) Wegen des Verschmelzens der ersten Zelle der vorhergehenden Zeile mit der ersten Zelle in dieser Zeile können hier maximal drei Elemente des Typs `entry` angegeben werden. Ein Blick auf TABELLE 6.5 zeigt jedoch, dass sich die nächste Zelle in dieser Zeile über die restlichen Spalten der Tabelle erstreckt (engl. SPANNING). Um dies zu erreichen, können zu einem `entry`-Element die Attribute `namest` und `nameend` angegeben werden. Die Werte dieser Attribute müssen jeweils einem Spaltenbezeichner entsprechen, der im Tabellenkopf in einem Attribut `colname` eines Elements vom Typ `colspec` festgelegt wurde.

Die CALS-Tabellen können relativ einfach in eigene Projekte integriert werden. BEISPIEL 6.10 zeigt ein XML-Dokument mit internem DTD-Subset, das genau dies leistet.

► **Beispiel 6.10: Einbindung von CALS-Tabellen in eigene Projekte**

```
<!DOCTYPE MeinXML [
<!ELEMENT MeinXML (para*,table)>
<!ELEMENT para (#PCDATA)>
<!ELEMENT Warnung (#PCDATA)>
<!ELEMENT Hinweis (#PCDATA)>
<!ENTITY % tbl.entry.mdl  "(para|Warnung|Hinweis)*">

<!ENTITY % CALS
  PUBLIC "-//OASIS//DTD DocBook XML CALS Table Model V4.1.2//EN"
  "http://www.oasis-open.org/docbook/xml/4.1.2/calstblx.dtd">
%CALS;

]>
<MeinXML>
  <para>...</para>
  <para>....</para>
  <table>
    <tgroup cols="3">
      <tbody>
        <!--...-->
        <row>
          <entry>
            <Warnung>Angegebene Mengen streng beachten
            &#x2014; Garantieverlust droht!</Warnung>
          </entry>
          <entry>...</entry>
          <entry>...</entry>
        </row>
      </tbody>
    </tgroup>
  </table>
</MeinXML>
```

Das kleine Beispiel definiert eine eigene XML-Anwendung und bindet die CALS-Tabellen über den Mechanismus der externen Entities ein. Wir müssen eine Parameter-Entity (markiert durch das Prozentzeichen `%`) verwenden, da es sich um eine Entity handelt, die innerhalb der DTD verwendet werden soll. Die Entity `tbl.entry.mdl` dient dazu, das Inhaltsmodell des Elementtyps `entry` der CALS-Tabellen um unser selbstdefiniertes Markup `Warnung`, `Hinweis` bzw. `para` zu ergänzen.

Dies führt uns zum folgenden Abschnitt, in dem wir sehen werden, wie wir andere Teile der DocBook-DTD für eigene Projekte verwenden oder die DTD um eigenes Markup erweitern können.

6.4 Parametrisierung – Modularisierung

Nicht immer deckt die DocBook-DTD die eigenen Bedürfnisse ab. Sie ist durch ihre vielen Freiheitsgrade nicht in jedem Fall für einen maximal produktiven Einsatz geeignet.

Die DTD ist stark modular aufgebaut und erleichtert dadurch eben jene Modifikationen, die eine Anpassung an die lokalen Gegebenheiten erfordert. Im Folgenden werden wir den Aufbau der DocBook-DTD skizzieren und diejenigen Stellen genauer betrachten, die für typische Modifikationen relevant sind. Im Anschluss daran werden wir das Zusammenspiel anhand exemplarischer Modifikationen betrachten.

Aufbau der DocBook-DTD

Die DocBook-DTD ist sowohl logisch als auch auf der Dateiebene stark modularisiert. Die Teilbereiche für die diversen Entities, Notationen sowie hierarchische und inhaltsbezogene Elementtypen sind jeweils in eigene Dateien ausgelagert. Auf syntaktischer Ebene macht die DocBook-DTD intensiven Gebrauch von CONDITIONAL SECTIONS – speziellen Bereichen, die bei Bedarf in die DTD aufgenommen oder ignoriert werden. Wir haben bereits in Lerneinheit 4 kennen gelernt, wie mit CONDITIONAL SECTIONS und PARAMETER-ENTITIES DTDs modularisiert werden können, so dass sie sich leicht für verschiedene Zwecke anpassen lassen. Dabei werden inhaltlich zusammenhängende Teile zu Blöcken zusammengefasst, so dass deren Inhaltsmodelle wiederverwendet, angepasst oder weggelassen werden können. Zum Einbinden bzw. Weglassen dieser bedingten Bereiche kann man Parameter-Entities mit den Schlüsselwörtern `INCLUDE` bzw. `IGNORE` verwenden.

▶ **Anmerkung:**
Wir beziehen uns im Folgenden auf die XML-Fassung der DocBook-DTD. Die SGML-Fassung weicht z.B. bei der Benennung der Dateien sowie an einigen weiteren Stellen vom hier vorgestellten Muster ab. Der prinzipielle Aufbau sowie die zur Verfügung stehenden Sprachmittel sind in beiden Varianten jedoch im Wesentlichen gleich.

Dateien der DocBook-DTD

TABELLE 6.6 listet die wichtigsten Dateien der DocBook-DTD auf.

Dateiname	Beschreibung
`docbookx.dtd`	Die zentrale Steuerdatei. Sie enthält keine Elementtyp-Deklarationen, sondern bindet lediglich andere Module ein.
`dbhierx.mod`	Definiert die hierarchische Struktur der Elemente in einem DocBook-Dokument und entsprechende Parameter-Entities für Attribute und Inhaltsmodelle.
`dbpoolx.mod`	Der Pool enthält die Definitionen für inhaltsorientiertes Markup.
`dbcent.mod`	Das Modul Character. Entities ist für die Einbindung der vorgebenen Entity-Deklarationen für Sonderzeichen zuständig.

Tabelle 6.6: DocBook-Dateien

▶ **Anmerkung:**

Bei allen folgenden Modifikationen werden die Distributionsdateien der DocBook-DTD NICHT verändert. Veränderungen finden ausschließlich auf der logischen Ebene statt.

Wichtige Parameter-Entities

Praktisch alle relevanten Aspekte sind in der DocBook-DTD in Parameter-Entities zusammengefasst. Eine (fast) durchgängige Namenskonvention erleichtert uns im Folgenden die Identifikation der jeweils betroffenen Entities. TABELLE 6.7 zeigt eine Übersicht über häufig verwendete Präfixe und Suffixe in Element- und Attributnamen in der DocBook-DTD. In Anlehnung an die Platzhalterzeichen vieler Kommandozeileninterpreter wird hier das Zeichen * als Platzhalter für eine Folge beliebiger Zeichen verwendet, die in Element- und Attributnamen in XML erlaubt sind. So steht `*.module` beispielsweise für alle Namen, die mit der Zeichenfolge `.module` enden.

Namenskonvention	Beschreibung
`*.module`	Zum Beispiel emphasis.module, aber auch `dbpool.module`; Diese Entities dienen als „Schalter", um optionale Module der DocBook-DTD einzubinden. Jede zu einem solchen Modul gehörige Elementtyp-Deklaration und Attributdeklaration ist durch eine Conditional Section eingeschlossen; als Bedingung fungiert die Deklaration des Entitys als Zeichenkette „INCLUDE" oder „IGNORE" (siehe Beispiel unten).
`*.class`	Zum Beispiel `list.class`; fasst Elemente gleichen oder ähnlichen Typs zusammen.
`*.mix`	Zum Beispiel qandaset.mix; fasst im Allgemeinen Klassen von Elementen zusammen, die häufig gemeinsam Teil eines Inhaltsmodells sind.
`*.element` bzw. `*.attlist`	Zum Beispiel `label.element` bzw. `label.attlist`; markieren die Element- bzw. Attributdeklaration.
`local.*`	Zum Beispiel `local.label.attrib` oder auch `local.word.char.mix`; dies sind lokale Entities, die der gezielten Erweiterung bestehender Inhaltsmodelle bzw. Attribut-Definitionen dienen und die in der DocBook-Distribution als leere Zeichenkette deklariert werden.

Tabelle 6.7: Übersicht Parameter-Entities

Der in BEISPIEL 6.11 angeführte Ausschnitt der DocBook-DTD zeigt die Deklarationen für den Elementtyp `label`. Gut zu erkennen ist die Einbettung der einzelnen Teile in Conditional Sections, die von einer unmittelbar vorher deklarierten Entity abhängen. Das Inhaltsmodell und die Attribute werden mit Hilfe mehrfach verwendeter Entities angegeben.

► **Beispiel 6.11: DocBook-DTD: Das Modul label**

```
<!ENTITY % label.module "INCLUDE">
<![ %label.module; [
  <!ENTITY % local.label.attrib "">
  <!ENTITY % label.role.attrib "%role.attrib;">

  <!ENTITY % label.element "INCLUDE">
  <![ %label.element; [
    <!ELEMENT label (%word.char.mix;)*>
  <!--end of label.element-->]] >

  <!ENTITY % label.attlist "INCLUDE">
  <![ %label.attlist; [
    <!ATTLIST label
      %common.attrib;
      %label.role.attrib;
      %local.label.attrib;
    >
  <!--end of label.attlist-->]] >
<!--end of label.module-->]] >
```

Wir erinnern uns an dieser Stelle daran, dass die Reihenfolge von Deklarationen der gleichen Entity relevant ist und immer die erste Deklaration als gültig angesehen wird. Angewendet auf das obige Beispiel bedeutet dies, dass die Einbindung des Moduls LABEL unterbunden werden kann, wenn die Parameter-Entity `label.module` vor Erreichen von Zeile 1 bereits auf IGNORE gesetzt war. Dieses Prinzip werden wir im folgenden Abschnitt als Grundregel der DTD-Modifikation immer wieder finden.

Anpassungsschichten (Customization Layers)

Mit den Kenntnissen über den Aufbau der DocBook-DTD und die unterschiedlichen Gruppen von Parameter-Entities können wir uns den konkreten Modifikationen und ihrer Umsetzung zuwenden. Im Folgenden betrachten wir diese Anwendungsfälle:

- Restriktion
  Einschränkung der DocBook-DTD, z.B. zwecks Eingabe- bzw. Fehlerminimierung oder Umsetzung von internen Stilregeln
- Extension
  Erweiterung, z.B. bei Bedarf nach zusätzlichen Elementtypen oder Attributen
- Fragmentverwendung
  Wiederverwendung bestimmter Teile der DocBook-DTD

Das generelle Vorgehen ist immer sehr ähnlich. Wir deklarieren zunächst die in der DocBook-DTD verwendeten Parameter-Entities und binden anschließend die UNVERÄNDERTE FASSUNG der DocBook-DTD ein. Sofern wir beim Deklarieren der Parameter-Entities zusätzliche Bezeichner für Elementtypen oder Attribute eingeführt haben, folgen deren Deklarationen nun im Anschluss. Im Prinzip könnten diese natürlich auch zu Beginn deklariert werden, davon nehmen wir jedoch Abstand, da die empfohlene Reihenfolge die klare Trennung der Teilbereiche liefert und wir in der DocBook-DTD deklarierte Parameter-Entities so mitbenutzen können. Sofern die Abfolgen korrekt eingehalten werden, können nach dem gleichen Prinzip auch nacheinander die einzelnen Module der DTD eingebunden werden. Die Änderungen bleiben dadurch „näher" an den betroffenen Modulen, sind allerdings schwerer zu überblicken.

Sowohl bei den Einschränkungen als auch bei der Einführung neuer Elementtypen oder neuer Attribute besteht die Gefahr, dass für die Weiterverarbeitung bestehende Werkzeuge (etwa Stylesheets) entsprechend angepasst oder ersetzt werden müssen. Diese Anpassungen können sehr umfangreich ausfallen. Vor entsprechenden Änderungen sollten die Tragweite und die sich ergebenden Folgen abgeschätzt werden.

Einschränkung der DocBook-DTD

Am Beispiel des Moduls QANDASET („Question and Answer", z.B. für die Definition von Frage-/Antwort-Paaren einer Sammlung häufig gestellter Fragen, englisch „FAQ") wollen wir erarbeiten, wie Teile der DTD ausgeschlossen werden können.

Das Studium der DTD zeigt, dass die Elementtypen innerhalb der MARKED SECTION mit der Entity-Referenz `%qandset.content.module;` innerhalb von dbpoolx.mod deklariert werden. Das intuitive Vorgehen ist also, diese Parameter-Entity auf `IGNORE` zu setzen, um die Einbindung zu verhindern (siehe BEISPIEL 6.12).

www.oasis-open.org/docbook/xml/4.1.2/dbpoolx.mod

▶ Beispiel 6.12: Vorläufige Anpassung

```
<!DOCTYPE chapter [
<!ENTITY % qandset.content.module "IGNORE">
<!ENTITY % DOCBOOKDTD
  PUBLIC  "-//OASIS//DTD DocBook XML V4.1.2//EN"
  "http://www.oasis-open.org/docbook/xml/4.1.2/docbookx.dtd" >
%DOCBOOKDTD;
]>
<chapter>...</chapter>
```

Eine derart modifizierte DTD ist jedoch äußerst „unsauber", da der Elementtyp `qandaset` weiterhin im Inhaltsmodell einiger Elementtypen erscheint, jedoch nicht definiert ist. Eine Überprüfung mittels `nsgmls -s -wall xml.dcl Testdok.xml` liefert daher:

```
nsgmls:Testdok.xml:5:0:W: generic identifier
   "qandaset" used in DTD but not defined
```

Es müssen also noch alle Stellen identifiziert werden, an denen auf Bestandteile des Moduls QANDASET Bezug genommen wird. Ein Blick in die DTD zeigt auf, dass nur innerhalb der Entity `compound.class` der Elementtyp `qandaset` aufgeführt wird. BEISPIEL 6.2 zeigt den betreffenden Ausschnitt aus der DocBook-DTD.

```
<!ENTITY % ebnf.block.hook "">
<!ENTITY % local.compound.class "">
<!ENTITY % compound.class
                    "msgset|procedure|sidebar|qandaset
                     %ebnf.block.hook;
                     %local.compound.class;">
```

Beispiel 6.2: Definition von compound.class in dbpoolx.mod

Die Aufnahme einer Deklaration von `compound.class` ohne den entsprechenden Elementtyp in den ersten Teil unserer DTD ergibt leider immer noch nicht das gewünschte Resultat, da Parameter-Entities nur verwendet werden können, wenn sie bereits vorher deklariert wurden. Im konkreten Beispiel können die als Platzhalter vorgesehenen Entities `local.compound.class` und `ebnf.block.hook` jedoch vorab deklariert werden, um diese Grundbedingung zu erfüllen. Im Allgemeinen kann man jedoch nicht davon ausgehen, dass die Abhängigkeiten derart einfach aufzulösen sind, sondern muss eine Kaskade erwarten, insbesondere wenn mehrere Modifikationen durchgeführt werden sollen. Aus diesem Grund sind innerhalb der DocBook-DTD mehrere Platzhalter aufgenommen worden, an deren Stelle gezielt eigene Deklarationen eingefügt werden können.

Erweiterung der DocBook-DTD

Unter Umständen findet man auch in der umfangreichen Menge an Sprachmitteln in der DocBook-DTD keinen passenden Elementtyp für die spezifischen Erfordernisse, erwägt also die Einführung neuer Elementtypen oder Attribute. Auch die Einschränkung eines bestehenden Attributs auf bestimmte Werte ist aus Sicht der DTD als Erweiterung zu werten; allerdings schrumpft die Menge der möglichen gültigen, also validierbaren Dokumente.

Beispielhaft werden wir nun einen neuen Elementtyp `related` in die Klasse der Warnungen und Tipps (vgl. auch HINWEISE, KOMMENTARE, WARNUNGEN, TIPPS) einfügen, der z.B. Leseempfehlungen oder Hinweise auf ähnliche Gebiete aufnehmen könnte.

Die Deklaration der Parameter-Entity `admon.class` enthält die für Erweiterungen vorgesehene Entity `local.admon.class`. Diese werden wir entsprechend modifizieren, um den neuen Elementtyp in die Klasse einzufügen (vgl. BEISPIEL 6.13).

► Beispiel 6.13: Einen neuen Elementtyp einfügen

```
<!-- 1. Erweiterung der Klasse um 'related'-->
<!ENTITY % local.admon.class "| related">

<!-- 2. Einbinden der Standard-DTD-->
<!ENTITY % DOCBOOKDTD
  PUBLIC  "-//OASIS//DTD DocBook XML V4.1.2//EN"
  "http://www.oasis-open.org/docbook/xml/4.1.2/docbookx.dtd" >
%DOCBOOKDTD;

<!-- 3. Neue Deklarationen unter Verwendung der DocBook-Entities -->
<!ELEMENT related ((%admon.mix;)+)>
<!ATTLIST related %common.attrib;
                  %admon.role.attrib;
                  %local.admon.attrib;>
```

In Schritt 1 wird die in DocBook zur Erweiterung der Klasse ADMONITIONS dafür vorgesehene Parameter-Entity `local.admon.class` deklariert. Da diese Parameter-Entity zur Erweiterung der Parameter-Entity `admon.class` dient, wird dem Elementnamen der Konnektor | vorangestellt. Danach wird in Schritt 2 die DocBook-DTD durch eine Referenz auf eine entsprechende Parameter-Entity eingebunden. Im Schritt 3 wird dann der neue Elementtyp `related` zusammen mit seiner Attributliste deklariert.

Wiederverwendung von Fragmenten der DocBook-DTD

Häufig besteht der Wunsch, nur Teile der in DocBook verfügbaren Sprachmittel als eigenständige Module in eigenen Projekten verwenden zu wollen. Durch die Modularisierung der DTD wird dies teilweise erleichtert, wenngleich die Entflechtung und passende Deklaration der Parameter-Entities arbeitsaufwändig sein kann. Wir verzichten daher an dieser Stelle auf eine ausführliche, exemplarische Darstellung, da sie im Licht der Ausführung zu Restriktion und Extension keine substantiell neuen Erkenntnisse beitragen kann und wir am Beispiel der CALS-Tabelle das prinzipielle Vorgehen bereits kennen gelernt haben.

ZUSAMMENFASSUNG

In dieser Lerneinheit haben wir uns mit folgenden Aspekten von DocBook beschäftigt:

- Wir haben die grundlegende Unterteilung des DocBook-Vokabulars in strukturelles Markup und inhaltsbezogenes Markup nachvollzogen.
- Wir haben die wichtigsten Elementtypen des strukturellen Markup kennen gelernt, z.B. die Wurzelelemente für typische DocBook-Dokumente, Kapitel- oder Abschnittselemente sowie Absätze. Viele dieser Elementtypen können zur automatischen Verzeichnisgenerierung verwendet werden.
- Das inhaltsbezogene Markup dient nicht der Strukturierung des Textes, sondern wird für den eigentlichen Gegenstandsbereich des Textes verwendet. Wir haben verschiedene Elementtypen und Gegenstandsbereiche vorgestellt, z.B. Listen und Elementtypen für die GUI-Dokumentation. Dazu kamen allgemein verwendbare Elemente z.B. für Hinweise, Warnungen oder Tipps.
- Das DocBook-Vokabular liegt in physisch getrennter Form, d. h. als einzelne, zum Teil parametrisierbare Entities vor. Wir haben am Beispiel einer CALS-Tabelle gesehen, wie Entities so definiert werden, dass mit ihrer Hilfe ein eigenes Vokabular zusammengestellt werden kann.
- Entities bilden ein wichtiges Mittel zur Parametrisierung und Modularisierung der DocBook-DTD. Wir haben die DTD-Dateien und in ihnen verwendete Parameter-Entities angesprochen und drei Formen der DTD-Veränderung vorgestellt: Restriktion und Extension der DTD sowie die Verwendung von DTD-Fragmenten.

Aus verständlichen Gründen konnten wir nur Ausschnitte aus dem umfangreichen DocBook-Vokabular kennen lernen. Nachdem wir jedoch nun die strukturgebenden Teile sowie die dahinterliegenden Prinzipien kennen, sind wir in der Lage, eigene Dokumente zu erstellen und unser Repertoire an inhaltsorientiertem Markup der DocBook-DTD bedarfsorientiert zu erweitern und sogar durch Zufügungen an die eigenen Anforderungen anzupassen.

Die DocBook-DTD sowie die auf ihr aufsetzenden Werkzeuge werden kontinuierlich gepflegt und weiterentwickelt. Derzeit sind Module zu den Scalable Vector Graphics (SVG) sowie zu MathML in Vorbereitung.